CMPT 280

Topic 18: Graph Path Algorithms

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 18

Exercise 1

Number of Paths of a Specific Length

Recall from the readings that the algorithm for finding the number of walks of a specific length between nodes i and j is:

```
1  // Find the number of walks in the graph of length r between
2  // nodes i and j
3  Algoirthm numWalksij(i, j, r, A)
4  i, j - pair of nodes
5  r - desired walk length
6  A - adjacency matrix for the graph
7
8  Ar = the r-th power of A
9  return Ar(i,j)
```

a) What is the time complexity to multiply two $n \times n$ matrices?

b) What is the time complexity of the `numWalksij` algorithm?

Exercise 2

Path Existance

Recall from the readings the following algorithm for path existence:

```
1  // Is there a path from node i to node j?
2  Algoirthm isPath(i, j, A)
3  i, j - pair of nodes
4  A - adjacency matrix for the graph
5
6  B = A + A^2 + A^3 + A^4 + ... + A^(n-1)
7  return B(i,j) > 0
```

a) What is the time complexity of the `isPath` algorithm?

b) Modify the algorithm to return path existence for **all pairs** of nodes.

c) What is the time complexity of the algorithm in part b)?

Exercise 3

Warshall's Algorithm

Recall from the readings Warshall's algorithm for path existence:

```
1   Algorthm pathExistanceWarshall(A)
2   A - adjacency matrix of a graph
3
4   P = A
5   for r=1 to n
6       for i = 1 to n
7           for j = 1 to n
8               P(i,j) = max( P(i,r) * P(r,j), P(i,j) )
9
10  return P
```

What is the time complexity of the `pathExistenceWarshall` algorithm?

Exercise 4

a) True or false? Warshall's algorithm works on both directed and undirected graphs.

b) Would you use Warhsall's algorithm if you only need to know if a path existed between a single pair of nodes? Why or why not?

Exercise 5

Floyd's Algoirthm

Recall Floyd's algorithm from the readings:

```
1   Algorthm shortestPathsFloyd(W)
2   W - weight matrix of a weighted graph
3
4   D = W
5   for r=1 to n
6       for i = 1 to n
7           for j = 1 to n
8               D(i,j) = min( D(i,j), D(i,r) + D(r,j) )
9
10  return P
```

a) What is the time complexity of the `shortestPathsFloyd` algorithm?

b) Will Floyd's algorithm work if there are negative weights, but no negative cycles?

c) Will Floyd's algorithm work if there are negative cycles?

Dijkstra's Algoirthm

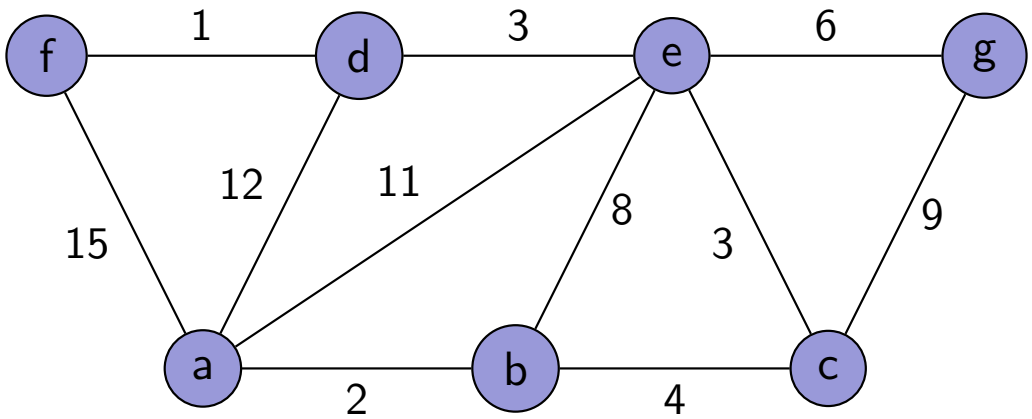
Recall Dijkstra's algorithm from the readings:

```
1   Algoirthm dijkstra(G, s)
2   G is a weighted graph with non-negative weights.
3   s is the start vertex.
4
5   Let V be the set of vertices in G.
6
7   For each v in V
8       v.tentativeDistance = infinity
9       v.visited = false
10      v.predecessorNode = null
11
12  s.tentativeDistance = 0
13
14  while there is an unvisited vetex
15      cur = the unvisited vertex with the smallest tentative distance.
16      cur.visisted = true
17
18      // update tentative distances for adjacent vertices if needed
19      // note that w(i,j) is the cost of the edge from $i$ to $j$.
20      For each z adjacent to cur
21          if (z is unvisited and z.tentativeDistance >
22                                  cur.tentativeDistance + w(cur,z) )
23              z.tentativeDistance = cur.tentativeDistance + w(cur,z)
24              z.predecessorNode = cur
```

Exercise 6

Dijkstra's Algoirthm

Trace through Dijkstra's algorithm manually for the following graph using node a as the start node:



For more practice on your own, try it using different nodes as the start node.

Exercise 7

Dijkstra's Algoirthm

```
1   Algoirthm dijkstra(G, s)
2   G is a weighted graph with non-negative weights.
3   s is the start vertex.
4
5   Let V be the set of vertices in G.
6
7   For each v in V
8       v.tentativeDistance = infinity
9       v.visited = false
10      v.predecessorNode = null
11
12  s.tentativeDistance = 0
13
14  while there is an unvisited vertex
15      cur = the unvisited vertex with the smallest tentative distance.
16      cur.visited = true
17
18      // update tentative distances for adjacent vertices if needed
19      // note that w(i,j) is the cost of the edge from i to j.
20      For each z adjacent to cur
21          if (z is unvisited and z.tentativeDistance >
22                                  cur.tentativeDistance + w(cur,z) )
23              z.tentativeDistance = cur.tentativeDistance + w(cur,z)
24              z.predecessorNode = cur
```

What is the time complexity of Dijkstra's algorithm?